JAPANESE KANJI SUGGESTION TOOL

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Computer Science

by

Sujata Dongre

December 2010

SAN JOSÉ STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Titled

JAPANESE KANJI SUGGESTION TOOL

by

Sujata Dongre

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Chris Pollett, Department of Computer Science 11/30/2010

Dr. Robert Chun, Department of Computer Science 11/30/2010

Dr. Mark Stamp, Department of Computer Science 11/30/2010

ABSTRACT

JAPANESE KANJI SUGGESTION TOOL

by Sujata Dongre

Many times, we can see that if we enter a misspelled search term in any of the search engines like Google, it will provide some help with "Did you mean:...." In my project, I am providing some suggestions for the wrong Japanese text entered by a user.

The Japanese language has three types of writing styles - hiragana, katakana and the most difficult, kanji. A single kanji symbol may be used to write one or more different compound words. From the point of view of the reader, kanji symbols are said to have one or more different "readings." Hence, sometimes it becomes very difficult to understand what the kanji symbols are and how to read them even if you know the Japanese language.There are different translation tools available nowadays that provide translation help. However, none of them provides any good suggestions for the incorrect Japanese words. In my project, I am developing a tool that will help users to correct their Japanese words by suggesting to them the correct word.

Keywords: Japanese, Hidden Markov Model, suggestions, natural language processing, find-as-you-type

ACKNOWLEDGEMENTS

I would like to thank my project advisor Dr. Chris Pollett for his support and guidance throughout this project. I would also like to thank my committee members, Dr. Robert Chun and Dr. Mark Stamp, for their valuable suggestions on my project work. I would also like to acknowledge my Japanese language professor, Inaba Sensei and my Japanese friend Tomoko san for their inputs in my project.

I am also very thankful to my parents, my husband and all my friends for their encouragement throughout my master's degree program.

Table of Contents

List of Figures

List of Tables

1. Introduction

In this multilingual world, everyone comes across different words in different languages. Japanese is a language spoken by over 130 million people in Japan and in Japanese emigrant communities [13]. There are several websites that provide online help with Japanese text which translates given Japanese text to any other language, and vice versa. The problem with these websites is if you enter an incorrect combination of Japanese words, then you will get either a meaningless translation or a "no search results found" message.

The Japanese language is written with a combination of three scripts: hiragana characters, katakana characters, and kanji symbols [12]. The total number of hiragana and katakana characters is around 90, while the total number of kanji characters is much larger but common-use kanji symbols number almost 2000. Like many other Asian languages, Japanese text is written without any spaces in between, which makes it hard to recognize the boundary between words. Therefore, it becomes necessary to figure out which words are present in the text and where these words begin and end. If you do not know which is the right combination of characters, you will probably end up getting some error message.

The "JUMAN" morphological analyzer [6] is one tool used for Japanese text segmentation. Rie Kubota Ando and Lillian Lee [10] tried an n-gram approach to the segmentation. Each of these approaches has some pros and cons [10]. I will explain later

how these analyzers and other algorithms for the Japanese language segmentation work. I will also look at how existing search engines provide help for the Japanese language.

My goal in developing a tool is to provide suggestions to people who have a very basic knowledge of the Japanese language. Even if they make mistakes while typing the Japanese characters, my tool will suggest to them the string that they might be looking for. This tool makes use of a Hidden Markov Model(HMM), which is trained on a corpus of Japanese text. The most commonly used Japanese corpus is the Tanaka Corpus [3]. The Tanaka Corpus was compiled by Professor Yasuhito Tanaka at Hyogo University and his students [3]. The Hidden Markov Model suggests the character with the highest probability for given input. Suppose a user enters the string as "$C_1C_2C_3$", then the HMM will suggest the string as "$C_1C_2C_4$" assuming the user might have entered C_3 incorrectly. Then it might also suggest " $C_5C_2C_3$" assuming user might have entered C_1 as the wrong character. Once the HMM suggests a string, it is given to the Nutch web crawler, which outputs the list of search results with the given suggested search string.

This project is divided into two main deliverables: The first deliverable was to develop a Japanese word segmenter using a HMM. This deliverable includes a HMM program that makes use of the Tanaka Corpus to provide probabilities for observations. Connected with the first deliverable, I also tested working of the HMM and Viterbi programs with different states and iterations. Then I compared results of HMM with the n-gram binary tree program. The second deliverable was to study and install the Nutch web crawler and incorporate existing HMM into the Nutch web crawler.

This report describes how programs were developed and experiments were conducted. It is structured as follows: Section 2 describes prior work about keyword suggestions in the Japanese text. Section 3 gives information about HMM for parsing Japanese text, followed by section 4 which provides the n-gram approach of text parsing. In section 5, I will explain the design and implementation of the project. Section 6 describes various experiments that are carried out and results of those experiments. At the end, section 7 gives the conclusion.

2. Prior work of keyword suggestions in Japanese

In Japanese language processing, the first key step is to find out accurate word segmentation because Japanese is written without delimiters between words. Many previously proposed segmentation methods for Japanese text make use of either a pre-existing lexicon or a substantial amount of pre-segmented training data. There are existing Japanese text analyzers like JUMAN morphological analyzer and TANGO algorithm.

2.1. JUMAN Morphological Analyzer

JUMAN is a morphological analyzer [6] for the Japanese language. It was developed by Yuji Matsumoto, Kazuma Takaoka, and Masayuki Asahara at the Matsumoto laboratory, Nara Institute of Science and Technology [8]. JUMAN is a rule-based Japanese morphological analyzer, where rules are represented as cost to lexical entry and cost to pairs of adjacent parts-of-speech, which are assigned manually. The probability of the occurrence of the word depends on the cost of a lexical entry, while a

connectivity cost of a pair of parts-of-speech reflects the probability of an adjacent occurrence of the pair [7]. However, this technique is a kind of labor-intensive and vulnerable to the unknown word problem. If the terms from the domain are not present in the lexicon, JUMAN will not work properly and hence it is not a very robust analyzer to work with. If the domain text changes, then re-estimation of costs will require much effort.

This analyzer provides word segmentation with some additional information such as their parts-of-speech. Similarly, there is also JTAG morphological analyzer [13].

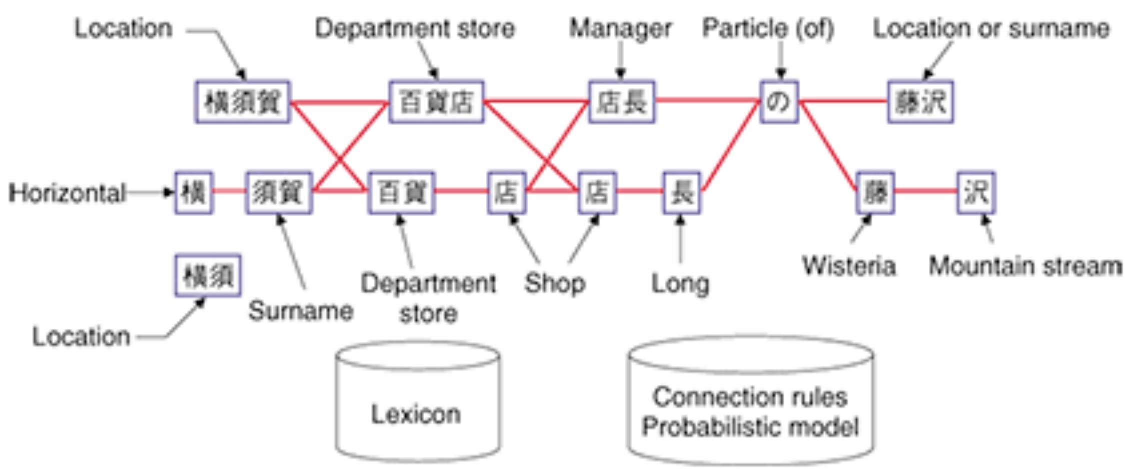


Figure1: Example of morphological analysis [13]

JUMAN is available on the internet for download and installation. Generally, morphological analysis is used to separate the text into the words, and these words are obtained from the dictionary. It also verifies if these words are connected to each other. If they are, then it evaluates the cost of word connection.

2.2. TANGO algorithm

"Tan-go bun-katsu(単語分割)" is Japanese for "word segmentation." This algorithm was developed by Rie Kubota Ando and Lillian Lee [10].

In this algorithm, the authors used the n-gram approach that provides counts for an unsegmented corpus to make segmentation decisions. The authors used a 4-gram approach to decide the word boundaries [10]. If the given sentence of eight kanji symbols says "$C_1C_2C_3C_4C_5C_6C_7C_8$" then the word boundary occurs in between C_4 and C_5. They pose series of questions like "is number of occurrences of $\{C_1C_2C_3C_4\}$ > number of occurrences of $\{C_2C_3C_4C_5\}$?" Each affirmative answer makes it more reasonable to place a segment boundary at the location under consideration [10].

In their paper, the authors present this simple TANGO algorithm that segments Japanese sequences into words based on statistics drawn from a large unsegmented corpus [10]. This algorithm is more robust than JUMAN analyzer, as it can be ported to any other domains and applications.

2.3. Search engines suggestions

I also studied how existing search engines like Google, Yahoo and Bing work for keyword suggestions in the Japanese language.

I will first briefly explain how Google provides suggestions. I think that Google search works better for suggestions in katakana, rather than suggestions in hiragana. Consider I start typing the query string for the word "you" in Japanese as "あなた". As a Japanese language beginner, I do not remember the Japanese characters and mistakenly

types “む” instead of “た”. So my query string becomes “あな む”, but the suggestions given by Google search engine are katakana words, although I am expecting it to show some simple suggestions as “あなた” (which is the most frequently used word in the Japanese language) [Figure 2].



Figure 2: Google search keyword suggestions for the word “あなむ”

Yahoo search works similarly to Google. Yahoo search also gives suggestions in katakana, but they are different from the ones suggested by Google [Figure 3].



Figure 3: Yahoo search keyword suggestions for the word “あなむ”

Bing search engine provides keyword suggestions that contain combinations of all the three letters such as “なむ”、”なむあ”、”あ” etc [Figure 4].



Figure 4: Bing search keyword suggestions for the word "あなむ"

3. Hidden Markov Model of text parsing

There are different approaches for text parsing and segmentation. The most commonly used approaches are Hidden Markov Model and N-gram. In my project, I am using the Hidden Markov Model approach to parse the Japanese text, as it maximizes the probability by considering all previous states. In the following section, I will explain briefly the working of these two models.

3.1. Markov model definition

The Markov Model is a finite automata model, which is used to find out what will be the next state of the model depending on the current state. Hence, to predict the future state, you do not have to look at the history of states of the model. e.g. Consider the

sentence "*The monster swallowed* ________". In this sentence, after the verb "*swallowed"*, it is less likely that the next word will be a verb again. Hence by looking at the current word which is "*swallowed*", I can say that the next word will be most likely some article such as "*a*", followed by a noun such as "*boy*". The purpose of the Markov Model is to find the next word with the highest probability. The transition probability is assigned to each path. It is better to follow the path with the highest probability to traverse from one state to the other.

3.2. Property of the Hidden Markov Model

In the HMM, the states that the model is passed through are unknown to us. We only know the observation sequence, probabilities of transitioning from one state to other, the emission probability for the current state, what are the chances that given observation will be generated.

HMM Example:

```
states = {'q', 'r'}
observations = {'eye', 'drops', 'off', 'shelf'}
start_probability = {'q': 1.0, 'r': 0.0}
transition_probability = {
        'q': {'q': 0.7, 'r': 0.3}
        'r': {'q': 0.4, 'r': 0.6}
}
emission_probability = {
        'q': {'eye': 0.4, 'drops': 0.3, 'off': 0.2, 'shelf': 0.2}
        'r': {'eye': 0.2, 'drops': 0.4, 'off': 0.1, 'shelf': 0.3}
}
```

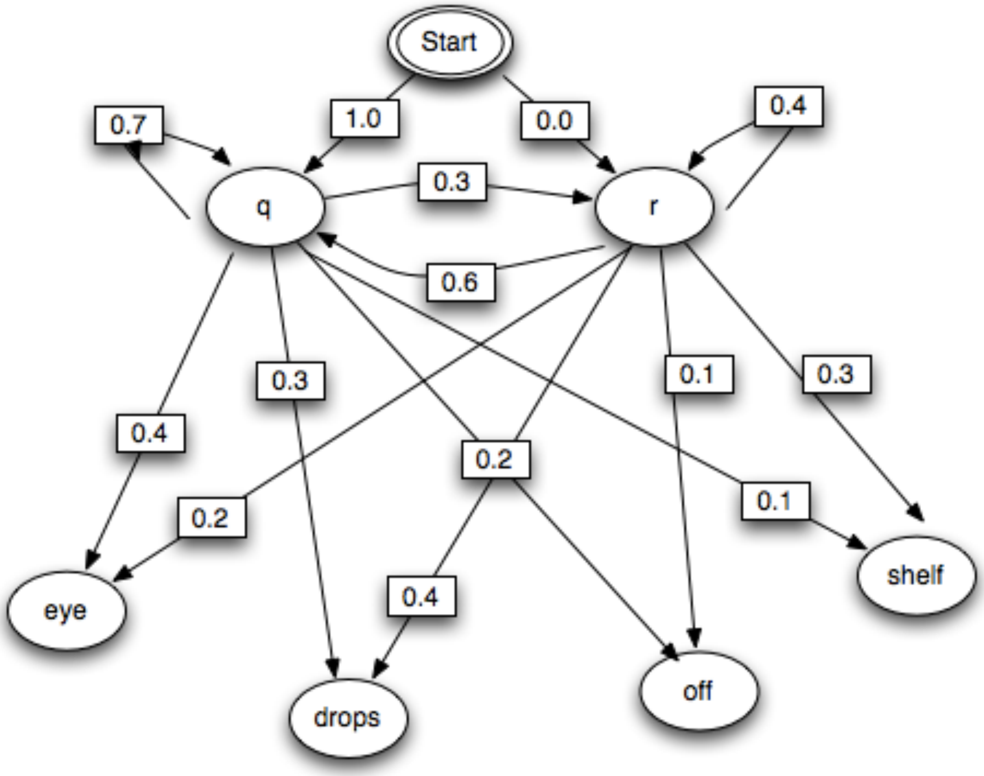


Figure 5: Example of the HMM [1]

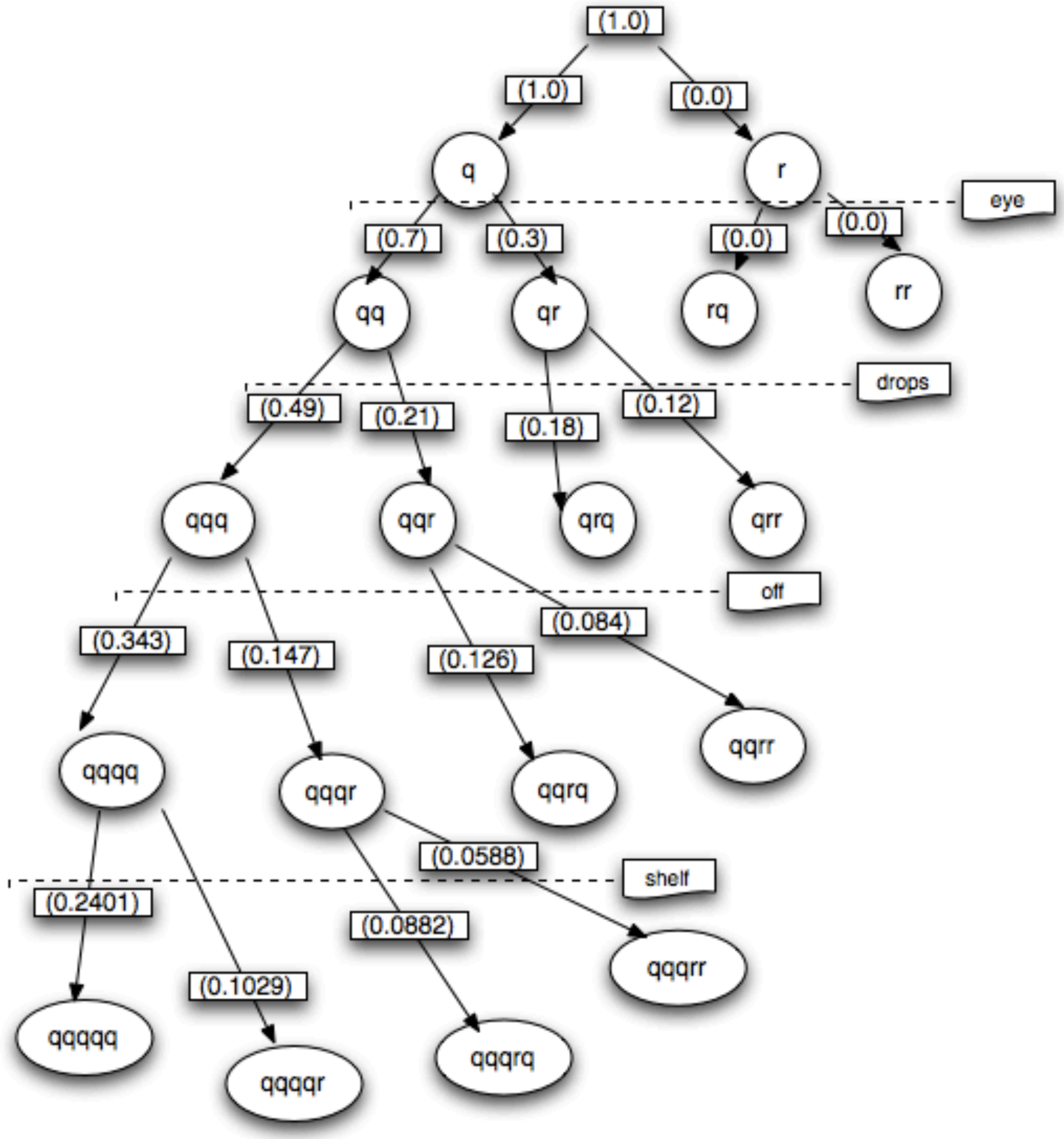


Figure 6: An alternate view of the search for the best path [2]

3.3. The Hidden Markov Model training

The sub-problems involved with the given Hidden Markov Model are as follows:

1. Given the Hidden Markov Model λ = (*A, B, π*) and a sequence of observations *O*, find *P(O|λ)*. Here, we want to determine the likelihood of the observed sequence *O*, given the model [11]. This problem can be solved using the Forward algorithm. The Forward algorithm is similar to the Viterbi algorithm except the purpose of the Forward algorithm is to find the sum of all possible ways to get to some end state. Thus, in the Forward algorithm, the probability of reaching some end state is the total product of all the probabilities of paths leading to that end state.

2. Given the Hidden Markov Model λ = (*A, B, π*) and an observation sequence *O,* find an optimal state sequence for the underlying Markov process. In other words, we want to uncover the hidden part of the Hidden Markov Model [11]. This problem can be solved using the Backward algorithm.

   Calculations for the Forward and the Backward algorithm are explained in tables below.

Time ticks	1	2	3	4	5
input	є	eye	eye drops	eye drops off	eye drops off shelf
Forward Probability of state q at timestamp t	1.0	0.28	0.0876	0.0149	0.0023
Forward Probability of state r at timestamp t	0.0	0.12	0.0444	0.007	0.0012
Total Probability	1.0	0.4	0.132	0.0219	**0.0035**

Table 1: Forward probabilities for "*eye drops off shelf*"

For given observation "*eye*", what is the total probability i.e., sum of the probabilities of all paths that one will end up in state "*q*" or in state "*r*"

Hence, in the above example,

$$\alpha_q(2) = [\alpha_q(1) \times (q \to q) \times (q_{eye})] + [\alpha_r(1) \times (r \to q) \times (r_{eye})]$$
$$\alpha_q(2) = [1.0 \times 0.7 \times 0.4] + [0.0 \times 0.6 \times 0.2]$$
$$\alpha_q(2) = 0.28$$

Where,

$\alpha_q(2)$ *: forward probability of state q at time 2*

$\alpha_q(1)$ *: forward probability of state q at time 1*

$q \to q$ *: transition probability from state q to next state q*

q_{eye} *: emission probability for observation eye from state q*

$\alpha_r(1)$: *forward probability of state r at time 1*

$r \rightarrow q$: *transition probability from state r to next state q*

r_{eye}: *emission probability for observation eye from state r*

Similarly, I can calculate the forward probability for other states and at different timestamps. As the sequence moves forward, it is called forward probabilities. We can also use the Backward algorithm instead of the Forward algorithm. The difference between the Backward algorithm and the Forward algorithm is, in the Backward algorithm I calculate the probability of sequence by working backward.

Time ticks	1	2	3	4	5
Input	eye drops off shelf	drops off shelf	off shelf	shelf	ϵ
Backward probability of state q at timestamp t	**0.0035**	0.0083	0.032	0.1	1
Backward probability of state r at timestamp t		0.0105	0.018	0.3	1

Table 2: Backward probabilities for "*eye drops off shelf*"

Hence, in the above example,

$$\beta_q(2) = [\beta_q(3) \times (q \text{ -> } q) \times (q_{drops})] + [\beta_r(3) \times (q \text{ -> } r) \times (q_{drops})]$$
$$\beta_q(2) = (0.032 \times 0.7 \times 0.3) + (0.018 \times 0.3 \times 0.3)$$
$$\beta_q(2) = 0.0083$$

Where,

$\beta_q(2)$: *backward probability of state q at time 2*

$\beta_q(3)$: *backward probability of state q at time 3*

$q \text{ -> } q$: *transition probability form state q to next state q*

q_{drops}: *emission probability for observation drops from state q*

$\beta_r(3)$: *Backward probability of state r at time 3*

$q \text{ -> } r$: *transition probability from state q to r*

3. Given an output sequence O, what HMM transition probabilities maximize the likelihood of the sequence? i.e., when we have the model and a set of all observations, how can I adjust parameters to increase the probability of the observations given in the model? This problem can be solved using the Forward-Backward algorithm, also known as the Baum-Welch algorithm. Using an initial parameter instantiation, the Forward-Backward algorithm iteratively re-estimates parameters and improves the probability that given observations are generated by the new parameters. I need to re-estimate three parameters such as initial state distribution, transition probabilities, and emission probabilities.

3.4. Working of the Hidden Markov Model

In the HMM, I know what the observable sequence is, but I do not know what states it has travelled to get this sequence. In the HMM, the states are not observable.

We know some probabilities that the model will pass through. The HMM works as any other finite state machine, having a set of hidden states, observable states, transition probabilities, initial state probabilities and output probabilities. The current state is not observable but each state produces an output with a certain probability. The HMM consists of a set of states as in any other finite automata model. e.g. states *{q,r}*, transition probabilities i.e., traveling from the state "*q"* to "*r"*, a set of observations such as string "*aabb*" and the emission probability of seeing a symbol "*b*" in state "*r*".

The problem with the HMM is how to find out the state sequence that best explains the given set of observations. Because it is possible to have more than one state sequences that will lead to the same observation sequence. This problem could be solved with the use of the Viterbi algorithm.

The Viterbi algorithm uses dynamic programming techniques. A dynamic programming can be viewed as the Forward algorithm, where "sum" is replaced by "max" [11]. Hence, in the Viterbi algorithm, the purpose is to find the probability of the best overall path [11]. Our example for the Viterbi algorithm for the given HMM is as follows:

Sentence: *eye drops off shelf*
Set of states = *{q, r}*
Set of observations = *{eye, drops, off, shelf}*

As the Viterbi algorithm follows the dynamic programming approach, it is required to store the best path and its probability for each possible state [1].

The purpose of using the HMM training is to find the HMM that maximizes the probability of the given observation sequence. If we know the observation sequence, we have to find the three parameters: initial state probabilities, transition probabilities, and emission probabilities. To achieve this I used the Forward-Backward/Baum-Welch algorithm. I followed the hill climbing approach. I started with some random HMM model and iteratively make small changes to the solution, improving it a little each time. When there is no more improvement possible, the HMM stops. I do not know what the model is. I can work out the probability of the observation sequence using some randomly chosen model. By looking at it, I can see which state transitions and emissions were probably used the most. By increasing the probability of those, I can choose a better revised model, which gives a higher probability to the observation sequence. Thus, I am following the process of maximization which is referred to as training the model. Hence, I will guess random transition probabilities from state *q* to state *r*. This is how I found the new re-estimated probability.

$$New\ P(q \text{ -> } r) = \alpha_q(t) \times \beta_r(t+1) \times (q \text{ -> } r)$$

Where,

$New\ P(q \text{ -> } r)$: *New estimated probability from state q to state r*

$\alpha_q(t)$: *forward probability from state q at timestamp t*

$\beta_r(t+1)$: *backward probability from state r at timestamp t+1*

$(q \text{ -> } r)$: *transition probability from state q to state r*

We will use this newly calculated probability for re-estimating start probabilities, transition probabilities and emission probabilities. Consider the following diagram for guessed HMM:

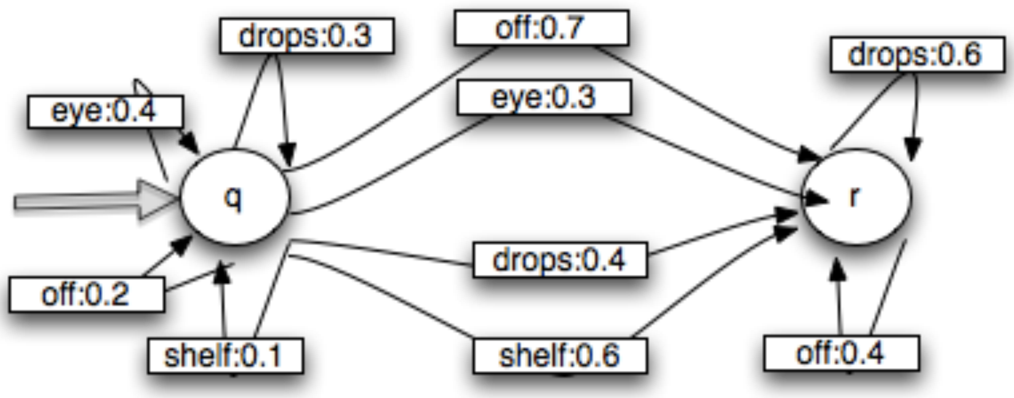


Figure 7: Guessed HMM

Thus, in the above example,

$$Total\ P(q \text{ -> } r) = \alpha_q(t1) \times (q \text{ -> } r) \times \beta_q(t2)$$
$$Total\ P(q \text{ -> } r) = 1.0 \times 0.3 \times 0.0105$$
$$Total\ P(q \text{ -> } r) = 0.00315$$

Where,

$Total\ P(q \text{ -> } r)$*: total probability from state q to state r*

$\alpha_q(t1)$ *: forward probability of state q at time t1*

$(q \text{ -> } r)$*: transition probability from state q to state r*

$\beta_q(t2)$ *: backward probability of state r at time t2*

Time ticks	t1	t2	t3	t4		
Input	eye	drops	off	shelf	**Total P**	**New P**
q -> r for observation eye	0.003	0	0	0	0.003	0.23
r -> r for observation drops	0	0.001	0	0	0.001	0.08
r -> r for observation off	0	0	0.005	0	0.005	0.38
r -> q for observation shelf	0	0	0	0.004	0.004	0.3
					0.013	

Table 3: New probabilities for "*eye drops off shelf*"

Similarly, I can calculate the new probability for other observations as well. After calculating all the total probabilities for remaining transitions, I calculate its total i.e., $\sum(Total\ P)$. This is nothing but the expected number of transitions from one state to the other state.

$$New\ P(q \text{ -> } r) = \frac{Total\ P(q \text{ -> } r)}{\sum_{1}^{n} Total\ P}$$

$$New\ P(q \rightarrow r) = \frac{0.003}{0.013}$$

$$New\ P(q \rightarrow r) = 0.23$$

Where,

$New\ P(q \rightarrow r)$ *: New probability from state q to state r*

$Total\ P(q \rightarrow r)$ *: total probability from state q to state r*

$\sum_{1}^{n} Total\ P$ *: Sum of all total probabilities*

My purpose of calculating total probabilities is to find out the new probability that will be greater than the previous probability. If I think that the difference between the new probability and the old probability is almost zero, I will stop iterating through the HMM and that will be my new re-estimated set of emission probability matrix.

4. N-gram approach to text parsing

N-gram is one of the methods that can be used for Japanese text parsing. It is used in the TANGO algorithm that I explained earlier.

4.1. N-gram definition and its use

N-gram is a probabilistic model, which predicts the next item in the sequence by looking at previous N-1 items. The n-gram model is one of the most important tools in speech and language processing. N-gram can be of size 1, which is referred to as a "unigram", size 2, which is a "bigram", size 3, which is a "trigram" and size 4 or more, which is simply called as an "n-gram" [4].

The n-gram approach is widely used in statistical natural language processing. For text parsing, words are modeled, such that each n-gram is composed of n words. In the above example of the Hidden Markov Model, the sentence is "eye drops off shelf", sequence of words in trigrams would be: "eye drops off", "drops off shelf". This approach is extremely useful in any task in which word identification is necessary in noisy, ambiguous input [9]. In speech recognition the input speech sounds are very confusable and many words have a similar sound. The n-gram approach is also a part of statistical machine translation process. If I have a set of potential rough translations, using n-gram I can tell which is the right translation that is being used most of the time.

4.2. N-gram model

N-gram model is a generalized Markov model. As seen in the previous section, the Markov property assumes that we can predict the future probabilities without knowing anything about history. In bigram, we assume that the probability of future words depends only on the previous words. Similarly, trigram will look into the past two words to predict the probability and n-gram, looks into the past N-1 words to predict future probabilities.

5. Design and implementation

In this section, I will explain how I implemented the project. Our project mainly involves two parts: HMM program and the Nutch web crawler.

5.1. Japanese language processing

Japanese is a language spoken by over 130 million people in Japan [13]. The Japanese language uses a combination of three writing styles: hiragana, katakana and kanji symbols. The hiragana is used for native Japanese words. The katakana is used for foreign words. The kanji symbols are a more pictorial representation of the word. Here is an example that uses all three scripts: kanji(K), hiragana(H) and katakana(KT)

私[K]は[H]サンノゼ[KT]大学[K]の[H]学生[K]です[H]。

This line can be translated in English as "I am a student of San Jose University." Kanji symbols are frequently used for nouns and the stems of the adjectives and verbs [5]. Hiragana is mostly used for endings of the verbs and for grammatical particles, while foreign borrowings are normally written using katakana.The most commonly used writing styles are kanji and hiragana.

In general in a Japanese-English dictionary, there are 150,00 entries of Japanese words. There are approximately 90 hiragana characters and 90 katakana characters. The total number of possible kanji symbols is disputed, but approximately 2,000 to 3,000 characters are used commonly in Japan [5]. Because of the huge number of kanji characters, in this project, I am going to map all kanji symbols as one symbol, "*".

The set of English characters is very small compared to the total number of Japanese characters. Hence, Japanese language cannot be encoded using only one byte. Thus, it has to be encoded as two-byte characters. To deal with this problem, many

encoding formats such as Shift-JIS, EUC and UTF8 are available to handle Japanese characters. In the project I decided to use "UTF8" character encoding to read corpus file, as it is the most commonly used encoding and works consistently for all operating systems and web browsers. Hiragana characters have a unicode range between 0x3040-0x309F [17]. Katakana characters have a unicode range between 0x30A0-0x30FF [18].

5.2. Hidden Markov Model program description

The HMM program is the heart of this project, which is written in Java using JDK1.6.

5.2.1. Number of iterations and observations

In my program, I used 100 iterations to build the final probability matrix of HMM. I decided on this number because after 100 iterations I could see convergence in the HMM probabilities. I could see a slight convergence at each iteration. First I used a total number of 50,000 observations for HMM. From the convergence of probabilities, I thought it would be better to use more observations so that I can see how HMM converges drastically from the first iteration to the last iteration. Hence I changed a total number of 100,000 observations.

5.2.2. Number of states

With more states, more time was required to run the HMM. I decided to build HMM for two, three, and four states and checked the difference in the probabilities. With

15 MB of the Tanaka Corpus, memory requirement to run HMM on four states will increase significantly. The final probability matrix for two, three, and four states is almost the same. Thus, I decided to use the HMM with two states to carry out all my experiments. All the resulting matrices for two and three states are listed in Appendix 1.

5.2.3. Japanese corpus

For carrying out all my experiments, I used Tanaka Corpus, which has 212,000 sentence pairs [3]. These sentence pairs are collected not only from textbooks used by Japanese students of English, but also from lines of songs, books etc.

The format of Tanaka Corpus is as follows:

- It contains pairs of lines starting with “A:” and “B:”, where sentence A is a Japanese sentence and its English translation. Sentence B contains a list of Japanese words found in sentence A.
- In sentence A, the Japanese sentence and its English translation are separated by [TAB]
- In sentence B, a line may contain a reading of kanji symbol in hiragana included in round brackets. In square brackets, you will find some IDs. These IDs indicate a sense number. If the word has multiple senses, then this sense number indicates which sense applies in the sentence. In the example below: 指す means “to point”/”to say”

  Consider the following example to understand the format:

  A: ＆という記号は、ａｎｄを指す。[TAB]The sign '&' stands for 'and'.#ID=1

  B: と言う{という}~ 記号~ は を 指す[03]~

As my project revolves only around Japanese text, I have written a program that will collect only Japanese sentences from the corpus file. Then for my experiments, I used this new corpus file.

5.2.4. The software

For my experiments with two states, three states, and four states in the HMM program, all the parameters for the number of states and number of iterations are handled from the Constants.java file. If in the future, anyone wants to run the HMM on different states, he/she should change the value in this Constants.java file.

The final matrix of start, transition and emission probabilities is stored as a serialized object in one file. All these file names are also stored in the Constants.java file. For the rest of the experiments, this serialized HMM is used so that execution time is saved. If the HMM is already built and stored, when a user enters a string to search, showing the corrected string will not take much of time, and the user will not have to wait longer to get the search results.

5.2.5. The GUI - the Nutch web crawler

The Nutch web crawler is used in my project as GUI. Nutch is an open source web crawler [14]. It maintains a database of pages and links. I implemented Nutch on the Windows 7 platform using Tomcat 5.5 amd JDK1.6 version. We also used cygwin to create and deploy jar files in Tomcat folder. We want to use Nutch so that when a user

enters a string, it will first call my HMM program to get the suggested string and use this corrected string to display the search results from the crawled pages.

After downloading and installing Nutch in my system, I made the following changes:

- To crawl Japanese websites, I used google.co.jp in crawl-urlfilter.txt file. This is my domain name to crawl.
- nutch-domain.xml is changed to add the agent name as google.
- nutch-site.xml file is changed to mention the searcher.dir property tag. This searcher.dir path is the path of my crawled dir

  Command to crawl:

bin/nutch crawl urls -dir crawljp -depth 3 -topN 10

Once the pages are crawled, I changed the existing search.html, search.jsp and nutch-1.0.jar file to incorporate the HMM program. I included my .class files of HMM program under the WEB-INF/classes folder of the Nutch project in Tomcat server. Also, a new jar file is created including the HMM .class files and the existing "nutch-1.0.jar" file is replaced with this new .jar file. It helped me to keep my code clean and easy to maintain. From the search.jsp page, I created an instance of the HMM program and called the right function to retrieve the suggested string.

Also, I implemented the instant search help functionality to the user using AJAX. Once a user starts to type in a query string, a suggested string is shown to him/her instantly.

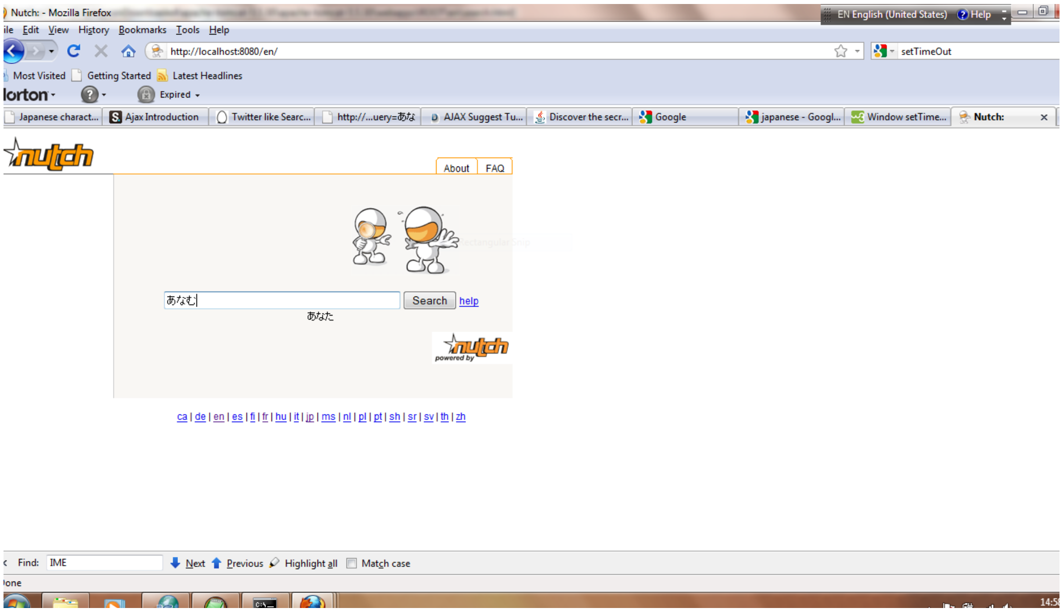


Figure 8: Instant query suggestion

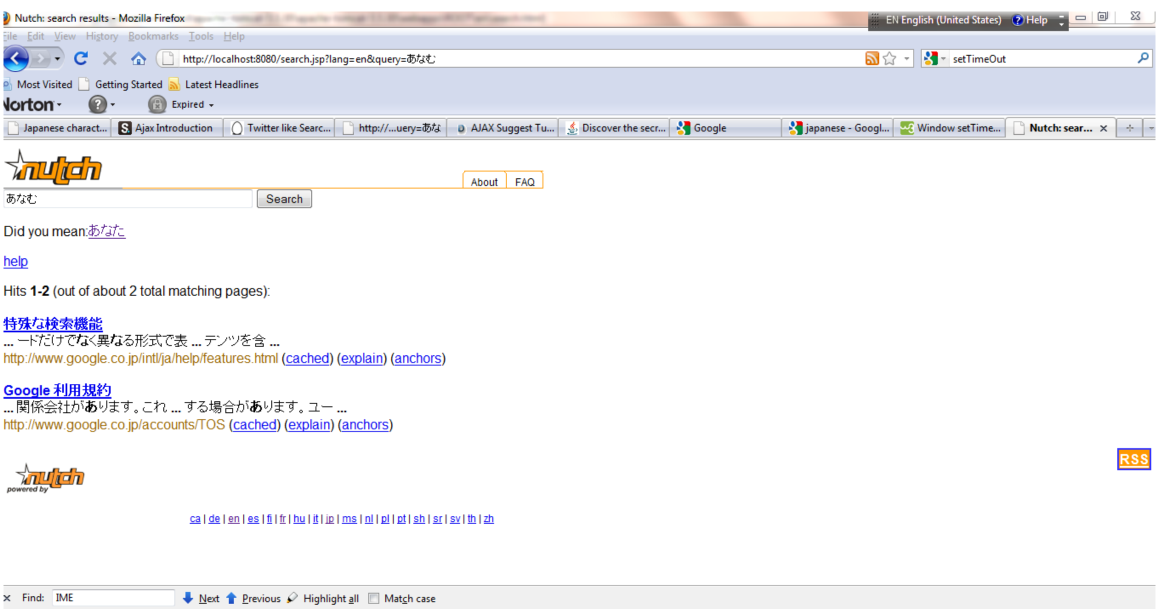


Figure 9: Link displayed for “Did you mean:”

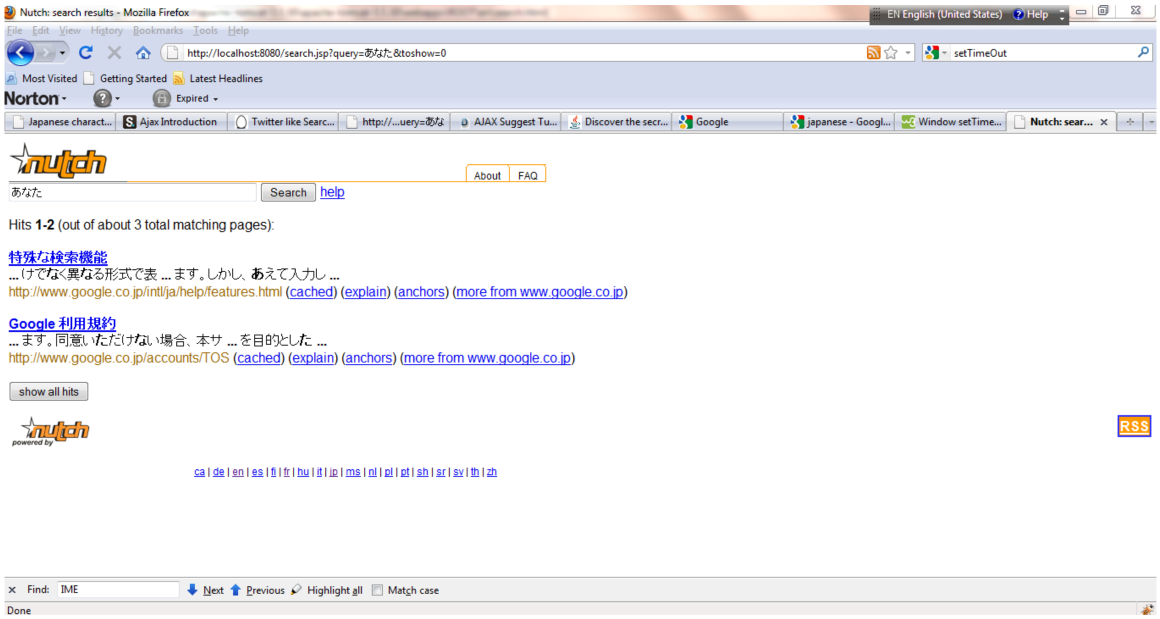


Figure 10: Search results after user clicks on "Did you mean" link

6. Experiments and Results

6.1. The HMM for English text

6.1.1. Goal

The main goal of testing the HMM program on English text is to understand how the HMM converges. For this experiment, I used "Brown Corpus" [16].

6.1.2. Results

My experiments were run with 100 iterations and 50,000 number of observations on "Brown Corpus" [16] with two, three, and four states. After 100 iterations, the matrix shows the letters with the highest probability are a, e, i, o, u and these are found in the first state which indicates that these letters appear more frequently at the beginning of the word. Hence, I could use HMM to distinguish between vowels and consonants. Also,

the significant change after 100 iterations is the observation "space" which is the character that has the highest probability among all 27 observations.

HMM for three and four states is not as helpful as it is for two states. In HMM two states, I can clearly see the separation between different states and characters. All the results for HMM English text with two and three states are listed in Appendix 2.

6.2. The HMM for Japanese text

6.2.1. Goal

Once testing the HMM for English text, I decided to run the same HMM program on Tanaka Corpus. My goal for this experiment is to check whether I can make a clear distinction between Japanese letters to identify space. If I can locate word boundaries, then it will help with Japanese text segmentation.

6.2.2. Results

I ran this experiment on HMM two states, three states and four states. It does not give any useful information for letters that have the highest probability, as I found in the English text. We can see that from the experimental results in the Appendix 1, the characters such as あ、い、う、お、で、の、は have slightly higher probabilities than the other characters.

I decided to use this HMM final probability matrix and Viterbi algorithm program along with the Tanaka Corpus. If a user enters a query string as "$C_1C_2C_3$", then using the HMM final probability matrix, the Viterbi program checks all the possible combinations of "$C_1C_2H_1$" to " $C_1C_2H_n$" and the string with the highest probability is

returned. Once I found the character with the highest probability from the Viterbi program, I verified that the string exists in the Tanaka Corpus. If no such string exists in the Tanaka Corpus, then I discarded that particular string and searched for the second highest probability string from the Viterbi program. Once the suggested string is also found in the Tanaka Corpus, that string is displayed on the screen for the user.

e.g. User enters string: あなむ

Viterbi program checks all the combinations such as “あなあ”、”あない”、。。。”あな＊” and then it returns the string with the highest probability. If this string also exists in the Tanaka corpus, then it will be displayed to the user. As in the above case, the suggested string is “あなた” (means “you” in English)

This experiment assumed the last character is wrong. Similarly, I did experiments changing the first character. For this project, I am assuming the query string of length three.

6.3. Tanaka corpus programs for parsing

6.3.1. Goal

I did a few experiments using Tanaka Corpus. In these experiments I followed the n-gram approach.

- The first experiment I did, involved iterating through each character in the corpus file and creating a binary search tree. A binary tree node consists of a key (word) and value (number of occurrences). Each word that is stored as a key in the binary tree node has a

length three. When any special character (characters other than hiragana, katakana and kanji symbols) are found, that special character is stored as “EOW” (End Of Word). After the experiment is completed, a binary tree with all the words and its number of occurrences is created based on the corpus file. When a user inputs a query string, this program looks into the binary search tree for the words starting with the user input and having the highest number of occurrences.

The goal of this experiment is to come up with suggestions for a possible next character for the given user query string.

- The second experiment is about finding word boundaries using the corpus file. In this experiment,the program I developed, is iterating through each and every letter in the corpus file, reading strings of length three. For each string of length three, its corresponding number of occurrences is also incremented. If the string ends with the special character (character other than hiragana, katakana and kanji symbols), then the number of occurrence variable is decremented, otherwise the number of occurrence variable is incremented. The aim to add or subtract the count is to find out words having negative number of occurrences.
- The third experiment is about creating a dictionary of Japanese words using corpus file. This experiment also stores words and the number of occurrences of each word. The difference between this experiment and previous experiments is previous experiment uses a string of length three. This experiment generates a bunch of Japanese words that can be used in security to create a dictionary of passwords in Japanese.

6.3.2. Results

After comparing the results of these experiments with the HMM program, I saw that these experiments give more accurate suggestions of strings than the HMM program. The binary tree program gives the list of the first three most common words that begin with the user entered string.

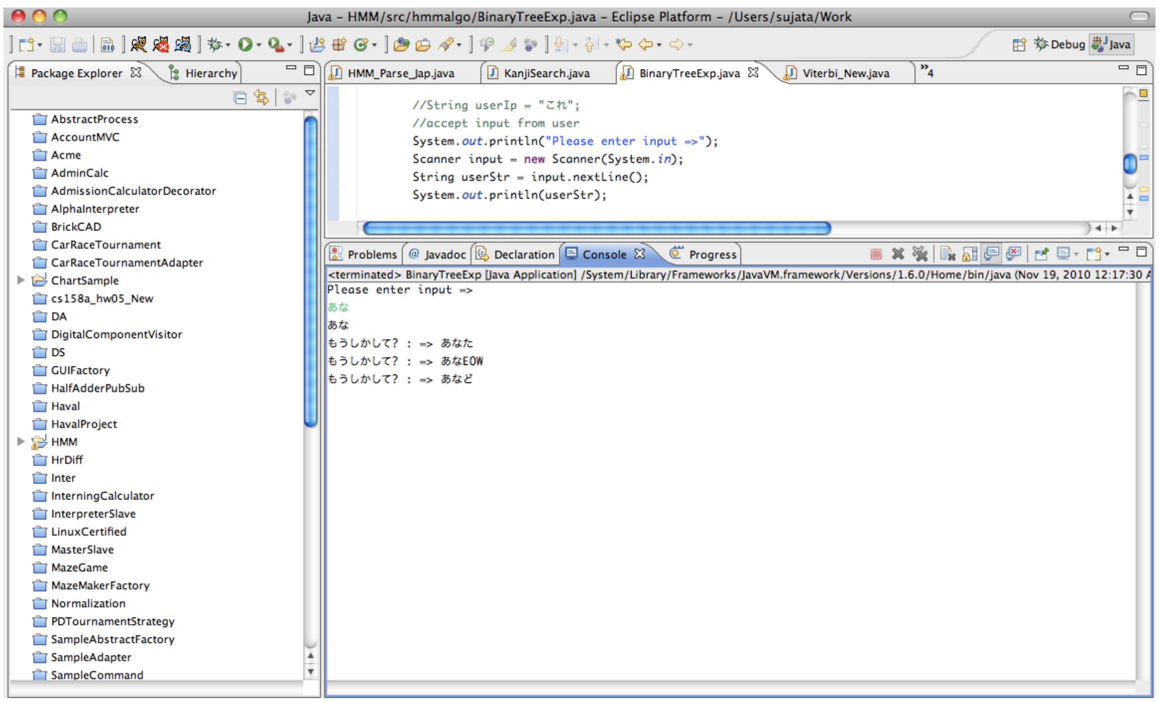


Figure 11: Binary Search Tree Experiment

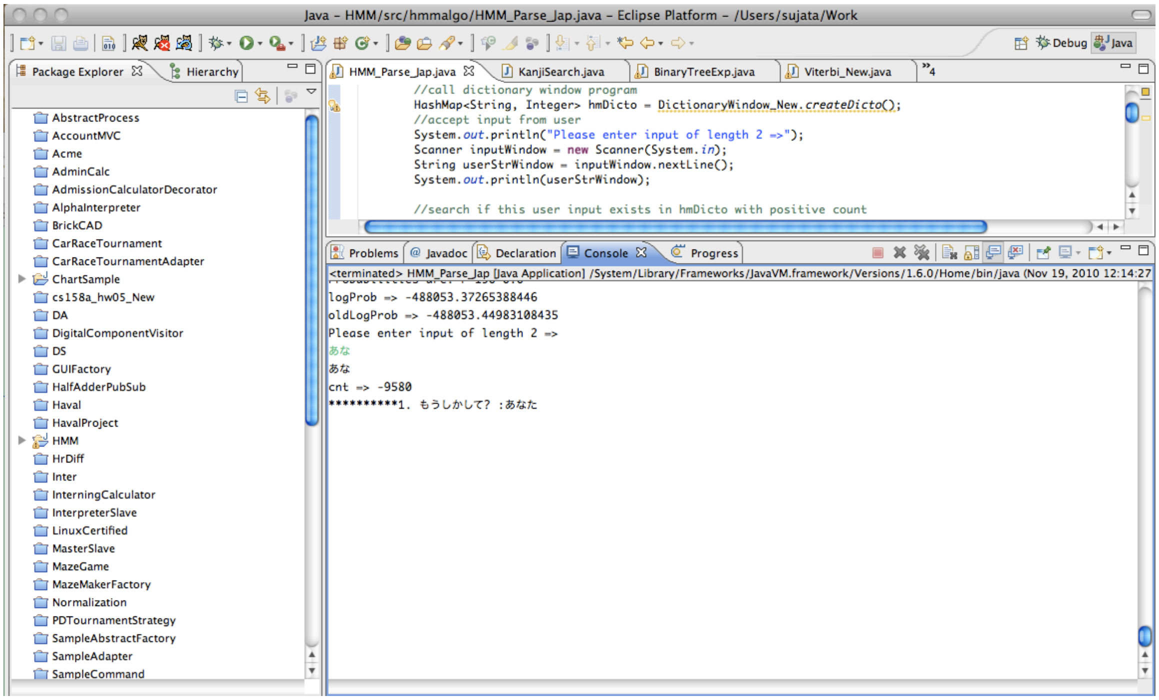


Figure 12: Dictionary experiment with window of length three

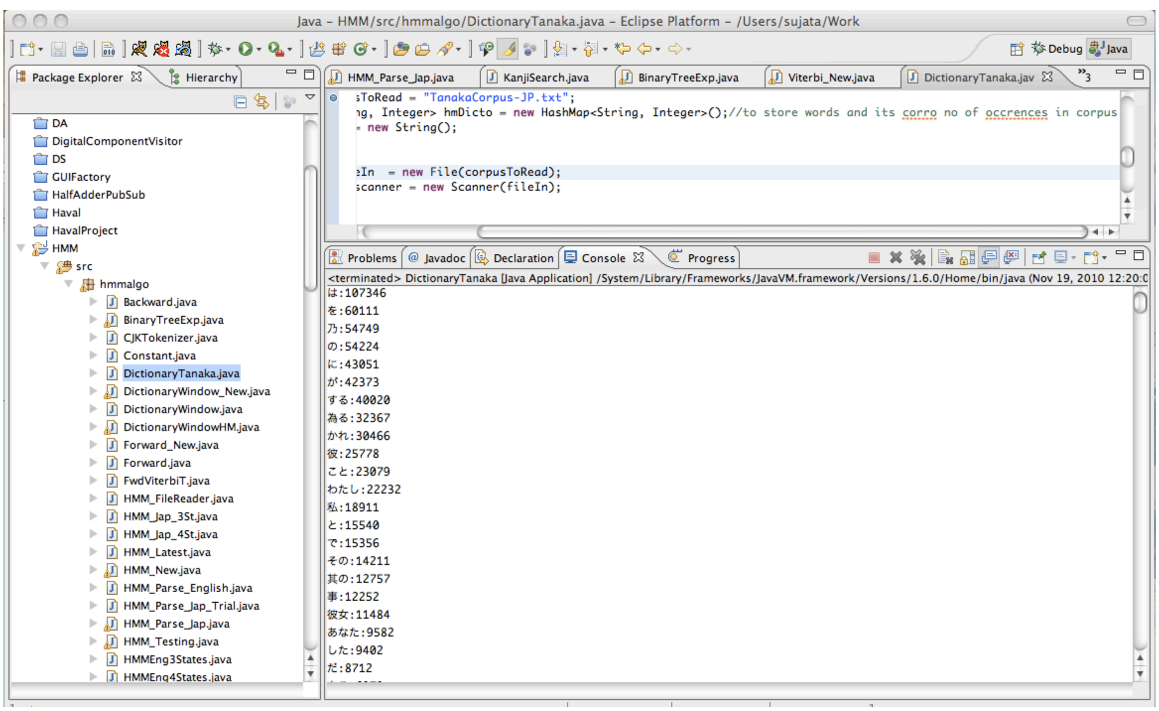


Figure 13: Experiment of creating Japanese word dictionary

6.4. Precision and recall

6.4.1. Goal

To verify the correctness of the outputs given by the HMM program, I decided to run an experiment of precision and recall on my program. The goal of this experiment is to evaluate how accurate the HMM program works. I received help from Seichiro Inaba, a Japanese professor at San José State University and my Japanese friend Tomoko Oshimo for conducting this experiment. I gave them 20 strings of length two and asked them to write down what is the most commonly used word starting with the two lettered string given to them.

In this experiment of checking the accuracy of HMM program, I followed two ways. I used the usual way of determining the likelihood of the observed sequence where I used the Forward algorithm to determine the probability. I also used another way of calculating the likelihood of sequence which is the Viterbi algorithm. All the results are recorded in the table below.

6.4.2. Results

Two letters string	Inaba Sensei	Tomoko san	Viterbi	HMM Forward	Binary tree output	Google search	Yahoo! search	Bing search
あな	あなた	あなた	あなた	あなた	あなた	あな／アナスイ	ANA ホームページ	ANA ホームページ

Two letters string	Inaba Sensei	Tomoko san	Viterbi	HMM Forward	Binary tree output	Google search	Yahoo! search	Bing search
でく	でくのぼう	でくの	でくの	でくた	でくわ／でくの	でくのぼう	デクスター	デクスター
あこ	あこがれ	あこがれ	あこの	あこい	あこが	アコム	アコム	アコム
だの	くだもの	だの？？	だのは	だので	No results found	ダノソ	ダノソ	ダノソ
れむ	クレーム	れむ？？	れむぢ	れむぢ	れむ EOW	レム睡眠	レム睡眠	レム睡眠
はこ	はこぶ	はこいりむすめ	はこな	はこた	はこ EOW	はこ boon	はこ boon	箱

Two letters string	Inaba Sensei	Tomoko san	Viterbi	HMM Forward	Binary tree output	Google search	Yahoo! search	Bing search
これ	これまで	これから	これな	これた	これ EOW／これら	コレガ	No suggestions	コレステロール
ドク	ドクター	ドクター	ドクロ	ドクロ	ドクロ	グエンドク	No suggestions	ドクター
ホテ	ホテル	ホテル	ホテル	ホテル	ホテル	ホテル	ホテル	ホテル
テレ	テレビ	テレビ	テレカ	テレカ	テレビ	テレビ台	テレビ	テレビ
買に	買いに行く	買に？？	買い	買い	No results found	No suggestions	買い物	No suggestions
言す	言います	言す？？	言い	言な	No results found	No suggestions	No suggestions	言いまわす

Two letters string	Inaba Sensei	Tomoko san	Viterbi	HMM Forward	Binary tree output	Google search	Yahoo! search	Bing search
上言に	上に言う	上言に？？	上言う	上言う	No results found	No suggestions	No suggestions	No suggestions
でぶ	でぶ	でぶっちょ	でぶら	でぶら	No results found	でぶにゃん	でぶや	debug assertion failed
わた	わたし	わたし	わたの	わたの	わたし	わた飴の日記	渡辺麻友	渡辺麻友
くに	くにく	くにがら	くにな	くにな	くに EOW／くにさ	ステーキくに	国生さゆり	国仲涼子
とて	とても	とても	とても	とてな	とても	とてノ	とり	とてつもなく
ここ	ここ	こころ	ここな	ここな	ここ EOW／こころ	ここカラダ	ココス	ココス

Two letters string	Inaba Sensei	Tomoko san	Viterbi	HMM Forward	Binary tree output	Google search	Yahoo! search	Bing search
もど	もどる	もどる	もどの	もどの	もどっ／もどり	もどリッチ	求索	戻れない
スリッ	スリッパ	スリッパ	スリッパ	スリッパ	スリッ	スリッパ	スリッパ	スリッパ

Table 4: Precision and recall experiment results

Precision is used as a measure of exactness whereas recall is a measure of completeness [15]. The program is more precise if it gives less number of irrelevant results. If the precision score is 1.0 means that the results returned from the program are all correct. And the recall score of 1.0 means that the program returns all the correct strings for all input strings.

For some of input strings, we can see either the input is changed to katakana word or remain unchanged or the string is completely converted to some different word. e.g. Consider a response given from Sensei and Tomoko for words such as " れむ" or " だの". In these cases, we can say that even native Japanese speakers got confused with the correct sequence of Japanese characters.

From the above results, I can calculate the precision and recall values for all, binary tree program, Viterbi, HMM and the search engines.

Precision = Number of correct results/ Number of returned results [20]

Precision for the Viterbi = 8/20 = 0.4

Precision for the HMM (Forward) =8/20 = 0.4

Precision for a binary tree = 8/15 = 0.53

Precision for the Google search = 4/17 = 0.23

Recall = Number of correct results/ Number of results that should have been returned[20]

Recall for the Viterbi = 8/20 = 0.4

Precision for the HMM (Forward) =8/20 = 0.4

Recall for a binary tree = 8/20 = 0.4

Recall for the Google search = 4/20 = 0.2

Hence, from the above experiment, I concluded that the binary tree program gives more relevant results as compared to HMM because the precision value of HMM is less than the precision value of the binary tree. With a set of 20 strings, there is no difference between the precision and recall values for HMM using the Forward algorithm and HMM using the Viterbi algorithm. However, the recall value is same in the HMM and a binary tree programs. The Google search has the lowest precision and recall.

7. Conclusion

The HMM technique and a binary tree are used to find the correct string with the highest probability. The most difficult part in the development of a HMM program is

handling large number of observation set for Japanese text and generating the start and transition probabilities randomly. To handle this large observation set, I mapped all kanji characters to one symbol. In the future, I can modify the existing HMM code, to recognize all kanji symbols and output suggestions for different kanji symbols. To generate probabilities randomly, I used Random class from Java.

The experimental results of precision and recall demonstrate that the n-gram approach of binary search tree is more reliable than the HMM approach. The binary tree program was written with an iteration window of size three based on Tanaka Corpus. Hence, if the user enters a wrong set of kanji and hiragana characters, a binary tree program fails to return any results, as no such string exists in the corpus file giving a precision of 0.53. The HMM program gives results for any characters typed by the user but those results are sometimes irrelevant. Hence, the precision of the HMM program is lower than the precision of a binary tree program. The search engines such as Google, Yahoo and Bing also suggests keywords but most of the time they are irrelevant. In the future, I can modify the existing binary search tree program by changing the iteration window size and running it more than one corpus file for Japanese text.

With thousands of online translation tools and search engines, none of the tools are efficient enough to provide suggestions for the wrong Japanese word. While learning the Japanese language or reading Japanese text, I always feel that the translation tool or the search engine should suggest a better word. The motive behind the development of

this tool is to encourage Japanese language learners to get a quick grasp of Japanese words.

8. References

[1] Viterbi Algorithm. Retrieved November 23, 2010, from http://en.wikipedia.org/wiki/Viterbi_algorithm

[2] Statistical Language Learning. Eugene Charniak. MIT Press.

[3] The Tanaka Corpus. Retrieved November 23, 2010, from http://www.csse.monash.edu.au/~jwb/tanakacorpus.html

[4] n-gram. Retrieved November 23, 2010, from http://en.wikipedia.org/wiki/N-gram

[5] Kanji. Retrieved November 23, 2010, from http://en.wikipedia.org/wiki/Kanji

[6] JUMAN information. Retrieved November 23, 2010, from http://www-lab25.kuee.kyoto-u.ac.jp/nl-resource/juman-eng.html

[7] Koichi Takeuchi, Yuji Matsumoto, HMM Parameter Learning for Japanese Morphological Analyzer. Retrieved November 23, 2010, from http://www.aclweb.org/anthology/Y/Y95/Y95-1022.pdf

[8] Chasen Morphological Analyzer User's Manual. Retrieved November 23, 2010, from http://sourceforge.jp/projects/chasen-legacy/docs/chasen-2.4.0-manual-en.pdf/en/1/chasen-2.4.0-manual-en.pdf.pdf

[9] N-GRAMS. Retrieved November 23, 2010, from http://www.mit.edu/~6.863/spring2010/readings/ngrampages.pdf

[10] Rie Kubota Ando, Lillian Lee, Mostly-Unsupervised Statistical Segmentation of Japanese Kanji Sequences. Retrieved November 23, 2010, from http://www.cs.cornell.edu/home/llee/papers/segmentjnle.pdf

[11] Dr. Mark Stamp, A Revealing Introduction to Hidden Markov Models. Retrieved November 23, 2010 from http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf

[12] Japanese. Retrieved November 23, 2010, from http://en.wikipedia.org/wiki/Japanese_language

[13] Kenji Imamura, Kuniko Saito, and Hisako Asano, Basic Japanese Text Analysis Technology as a Platform for Knowledge Extraction. Retrieved November 23, 2010 from https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr200809sf4.pdf&mode=show_pdf

[14] Nutch Tutorial. Retrieved November 23, 2010 from http://nutch.sourceforge.net/docs/en/tutorial.html

[15] Precision and recall. Retrieved November 23, 2010 from http://en.wikipedia.org/wiki/Precision_and_recall

[16] Brown Corpus Manual. Retrieved November 23, 2010 from http://icame.uib.no/brown/bcm.html

[17] Hiragana characters chart. Retrieved November 23, 2010 from http://www.unicode.org/charts/PDF/U3040.pdf

[18] Katakana characters chart. Retrieved November 23, 2010 from http://www.unicode.org/charts/PDF/U30A0.pdf

[19] Nutch wiki. Retrieved November 23, 2010 from http://nutch.apache.org/about.html

[20] Information Retrieval. Retrieved November 23, 2010 from http://en.wikipedia.org/wiki/Information_retrieval

Appendix 1: HMM Program results for the Japanese text

HMM parsing on Japanese text

No. of observations read = 50000
No. of states = 2
Language = Japanese
Emission matrix size = 191
Iteration = 100th

Start Probabilities

q	**r**
0.959001807619696	0.0409981923803042

Transition Probabilities

	q	**r**
q	0.500624298487603	0.499375701512407
r	0.500671848129107	0.499328151870889

Emission Probabilities

	q	**r**
0	0	0
1	0.255653541151235	0.255676715913046
2	0.0260044515787143	0.0259164712050492
3	0	0
4	0.0486462157973132	0.0483152884871585
5	0	0
6	0.0146813851378645	0.0146391457565325

	q	r
7	0	0
8	0.0049293470662525	0.00491082539372312
9	0	0
10	0.00509379159492166	0.00510642901915708
11	0.0202026272037206	0.0206788162917639
12	0.0248784717036502	0.0248424754045021
13	0.00725287483510252	0.00722738126321316
14	5.76E-04	5.84E-04
15	0.00992731298140807	0.0100332254326073
16	7.82E-04	7.78E-04
17	0.00562610081503685	0.0056942151155227
18	7.38E-04	7.42E-04
19	0.0108074077777545	0.0107128984124685
20	0.00129431050259396	0.00130575654254878
21	0.00721671234893996	0.00726363893481072
22	4.02E-04	3.98E-04
23	0.0253519882573592	0.025048621032086
24	0.00107574290294857	0.00108431156013009
25	0.0229336759511032	0.0231475267421183
26	0.00139614901103806	0.00140391720020146
27	0.00344682095440728	0.0034733523198612
28	2.59E-04	2.61E-04

	q	r
29	0.00663795377998909	0.00672242444938009
30	8.50E-04	8.71E-04
31	0.0401264924593157	0.0404755783940263
32	0.0124329828237568	0.0124475339114177
33	0.00415502103693522	0.00416515869081832
34	0	0
35	0.0206173685945098	0.020182876346921
36	0.00665456035493773	0.006665720719015
37	2.59E-04	2.61E-04
38	0.023689704540606	0.0239515924908802
39	0.0252272332381718	0.0254540785362201
40	0.0209122797500036	0.0192463338300259
41	0.00448457451533148	0.00447559288324437
42	0.0301914619424668	0.0304500908106449
43	0.0294863467211587	0.0295549242373167
44	1.19E-04	1.21E-04
45	0.00187650261246155	0.00188358189369584
46	0.035025600776364	0.0348555733760735
47	0.0314494275606899	0.0319124482358832
48	0.00348809467410415	0.00351207684966858
49	9.96E-05	1.00E-04
50	4.62E-04	4.58E-04

	q	r
51	5.18E-04	5.22E-04
52	0	0
53	1.40E-04	1.40E-04
54	5.38E-04	5.42E-04
55	4.00E-05	4.00E-05
56	0.00120152846479128	0.00119851557617856
57	8.98E-04	9.02E-04
58	0	0
59	9.55E-04	9.65E-04
60	4.19E-04	4.21E-04
61	6.02E-05	5.98E-05
62	0.0205973495744839	0.0202830549487243
63	0.00148312226740041	0.00147692879339099
64	3.79E-04	3.81E-04
65	0.00233812167925083	0.00238203043410955
66	0.00961550895854897	0.00974504850950515
67	6.19E-04	6.21E-04
68	0.00203867468764317	0.00204141050981677
69	3.99E-05	4.01E-05
70	1.20E-04	1.20E-04
71	0.00251061016645553	0.00252951548376449
72	0.00663661891901119	0.0066837093796643

	q	r
73	0.0114057132683051	0.0114347817469863
74	0.00967010201780335	0.00973036519686675
75	0.0189643777702905	0.0187961563429793
76	0.00987339881121645	0.0100872815134994
77	0.0028089040678766	0.00279118464483961
78	0	0
79	0.00356378777593715	0.00355634484429364
80	0	0
81	0	0
82	0.0181651721889989	0.0182356484560436
83	0.00854211981250269	0.00853821666422648
84	0	0
85	0	0
86	0	0
87	0	0
88	0	0
89	0	0
90	0	0
91	0	0
92	2.01E-05	1.99E-05
93	0	0
94	0	0

	q	r
95	0	0
96	0	0
97	1.19E-04	1.21E-04
98	0.00350317303937454	0.00353701223999516
99	6.79E-04	6.81E-04
100	0.002762682157663	0.00275742132997613
101	0	0
102	8.19E-04	8.21E-04
103	2.78E-04	2.82E-04
104	2.59E-04	2.61E-04
105	9.97E-05	1.00E-04
106	3.56E-04	3.64E-04
107	9.20E-04	9.20E-04
108	2.00E-04	2.00E-04
109	6.20E-04	6.20E-04
110	6.39E-04	6.41E-04
111	0.0015453360510733	0.00157476503015364
112	5.16E-04	5.24E-04
113	2.38E-04	2.42E-04
114	9.91E-05	1.01E-04
115	8.92E-04	9.08E-04
116	1.60E-04	1.60E-04

	q	r
117	7.24E-04	7.16E-04
118	1.80E-04	1.80E-04
119	0.00106065383538851	0.00105938690026331
120	0.00138701622616549	0.00137302057976056
121	0.00346101735249325	0.00345911855458977
122	2.37E-04	2.43E-04
123	7.98E-04	8.02E-04
124	2.00E-05	2.00E-05
125	2.61E-04	2.59E-04
126	0	0
127	0.00113543679827419	0.00114462087372748
128	3.80E-04	3.80E-04
129	4.99E-04	5.01E-04
130	0	0
131	0.00135025415272939	0.00136982597258023
132	2.39E-04	2.41E-04
133	0	0
134	0.00106887843925215	0.00109119409219051
135	2.79E-04	2.81E-04
136	0.0031346864937812	0.00314545326003017
137	0.0024849629091144	0.00251517680451934
138	3.18E-04	3.22E-04

	q	r
139	7.52E-04	7.68E-04
140	4.01E-05	3.99E-05
141	1.38E-04	1.42E-04
142	1.80E-04	1.80E-04
143	2.58E-04	2.62E-04
144	0.0010560585746586	0.00106399425645285
145	8.40E-04	8.40E-04
146	2.41E-04	2.39E-04
147	4.01E-04	4.00E-04
148	3.77E-04	3.83E-04
149	8.45E-04	8.35E-04
150	4.75E-04	4.85E-04
151	5.98E-04	6.02E-04
152	3.98E-05	4.02E-05
153	3.18E-04	3.22E-04
154	6.24E-04	6.16E-04
155	2.98E-04	3.02E-04
156	3.19E-04	3.21E-04
157	3.39E-04	3.41E-04
158	9.65E-04	9.55E-04
159	3.19E-04	3.21E-04
160	6.79E-04	6.81E-04

	q	r
161	9.21E-04	9.19E-04
162	1.59E-04	1.61E-04
163	4.59E-04	4.61E-04
164	1.39E-04	1.41E-04
165	7.42E-04	7.38E-04
166	8.03E-05	7.97E-05
167	3.59E-04	3.61E-04
168	2.00E-04	2.00E-04
169	0.00131540605036662	0.00132465891218842
170	0.0017868133633409	0.00177323999793018
171	0.0034978141473505	0.00354238523747827
172	8.38E-04	8.42E-04
173	6.02E-04	5.98E-04
174	0	0
175	1.39E-04	1.41E-04
176	0	0
177	0	0
178	0	0
179	0.00523848204864668	0.00520167838263065
180	0	0
181	1.99E-05	2.01E-05
182	3.81E-04	3.79E-04

	q	r
183	0	0
184	0	0
185	0	0
186	0	0
187	6.36E-04	6.44E-04
188	0.00707810042953468	0.00712224159308415
189	0	0
190	0	0

Log Prob => -252466.21402948367

No. of observations read = 50000
No. of states = 3
Language = Japanese
Emission matrix size = 191
Iteration = 100

Start Probabilities

q	r	s
0.999073535209081	9.26E-04	1.63E-40

Transition Probabilities

	q	r	s
q	0.0721456836517495	0.525792942773241	0.402061373575021
r	0.015507204602405	0.795935363455294	0.188557431942299
s	0.871370357994302	0.0155673871852324	0.11306225482047

Emission Probabilities

	q	r	s
0	0	0	0
1	4.95E-05	0.447183784076656	0.00730490429186804
2	0.0750625828636172	0.0018879778794974	0.0404963564466616
3	0	0	0
4	0.055859237706221	0.0166509314934374	0.124259191281704
5	0	0	0
6	1.15E-04	0.018560375450746	0.0187819026052506
7	0	0	0

	q	r	s
8	0.0024736016471003	9.78E-04	0.0176086504388835
9	0	0	0
10	1.45E-05	0.0089687822371504	6.68E-06
11	0.0422359840922002	0.0095102409212093	0.0275311221124488
12	0.00115608158163583	0.039113177373367	0.0109789371513374
13	7.18E-05	0.00493811324241329	0.0202902596690599
14	7.79E-16	4.14E-04	0.00158290777018252
15	0.0058193661827848	0.0103724217314838	0.0130463981837743
16	0.0027154663681333	6.78E-05	7.36E-04
17	0.0092434289843684	0.00127671981704282	0.0135726057533877
18	4.61E-08	3.49E-04	0.00248659670233503
19	5.36E-04	0.0183205716906812	0.00108741333204224
20	2.94E-06	0.00190932966668195	9.85E-04
21	0.0100433472343484	0.00284808041575846	0.0159419617633978
22	1.87E-04	5.95E-04	9.99E-05
23	1.48E-14	0.00178226338971655	0.111055132671909
24	3.73E-21	5.96E-04	0.00340329056000311
25	3.39E-28	0.0183329182073274	0.0579647103028998
26	3.54E-05	0.00218981971087112	6.81E-04
27	4.04E-23	2.34E-12	0.015886373642921
28	0.0010717620165509	5.38E-05	9.56E-17
29	0.00182578874311722	0.0110699771307572	3.39E-17

	q	r	s
30	0.0040004880677752	6.47E-06	1.70E-08
31	0.18622225290543	7.72E-04	1.19E-08
32	1.18E-04	0.0105535524396962	0.0294726001019325
33	0.0010470255933469	0.00467687003076349	0.00587158267174382
34	0	0	0
35	1.66E-19	6.27E-28	0.0936653244047042
36	0.0154660404660608	0.0058955531675707	1.63E-09
37	2.03E-28	3.90E-04	1.77E-04
38	0.111280579435487	8.91E-11	4.87E-10
39	0.0060194410962442	0.0423345088413811	1.05E-07
40	0.0182761081009938	0.0270359805865832	0.00370983303912165
41	0.00301585986711069	0.00674924074898842	1.15E-10
42	0.0132090115057985	0.0052811533466005	0.112454332168165
43	0.0012390945426299	0.0514926050862331	1.31E-06
44	2.64E-10	7.39E-05	3.58E-04
45	0.0015978702706968	9.77E-04	0.00451280504676832
46	7.70E-04	0.0612099101591007	2.77E-09
47	0.0027163479801558	0.0537107698171013	0.00268079887060554
48	0.0138656971027275	4.95E-04	0.00115200554804961
49	4.67E-04	5.26E-97	9.05E-59
50	3.87E-04	4.53E-04	5.51E-04
51	1.67E-04	5.99E-04	6.62E-04

	q	r	s
52	0	0	0
53	2.15E-04	1.13E-04	1.37E-04
54	1.05E-04	7.66E-04	3.78E-04
55	2.84E-47	1.61E-45	1.84E-04
56	1.38E-04	0.00206038235431289	3.24E-11
57	5.34E-06	6.88E-04	0.00233276427940296
58	0	0	0
59	1.32E-04	0.00163566865875866	1.12E-05
60	8.41E-05	7.08E-04	5.13E-09
61	2.80E-04	1.64E-87	9.28E-76
62	0.0780839970553065	0.00589523579938521	0.00172907372441659
63	9.62E-04	5.13E-04	0.00451103891347507
64	8.05E-15	5.35E-04	3.49E-04
65	1.59E-05	0.00113095037621496	0.00787001163060328
66	5.54E-04	0.0151935056710806	0.00426743354652492
67	0.0028964718599479	1.36E-47	2.76E-23
68	0.0018520723404992	0.00176862670838526	0.00293277113704503
69	1.87E-04	3.06E-32	1.08E-67
70	2.00E-04	1.14E-04	5.74E-05
71	0.0117727565920463	1.36E-64	2.30E-84
72	0.0110437766992533	0.00581296383218547	0.0045616984761095
73	0.0365630031504216	0.00632524193268217	1.03E-08

	q	r	s
74	2.60E-07	0.00946868373642041	0.0198373210386619
75	0.0791899063575108	0.00308443490942117	8.12E-04
76	8.46E-04	0.00314777662850644	0.0367798671265133
77	0.0079047489622625	0.00195018941648789	2.46E-18
78	0	0	0
79	2.75E-04	0.00301826131426467	0.00820184230553746
80	0	0	0
81	0	0	0
82	0.0013685645829742	0.0315193025907701	3.61E-12
83	0.026045103838904	0.00518549978425972	8.69E-05
84	0	0	0
85	0	0	0
86	0	0	0
87	0	0	0
88	0	0	0
89	0	0	0
90	0	0	0
91	0	0	0
92	4.92E-42	3.52E-05	1.11E-45
93	0	0	0
94	0	0	0
95	0	0	0

	q	r	s
96	0	0	0
97	5.61E-04	2.43E-106	1.13E-17
98	0.006827789714856	6.28E-09	0.00945140058046275
99	0.0031767755877433	9.20E-96	5.77E-13
100	0.0053169851290699	4.48E-34	0.00744676341605047
101	0	0	0
102	0.002123543060634	1.54E-41	0.00167793224207971
103	0.00130727971106086	1.99E-86	7.91E-07
104	6.22E-18	1.34E-42	0.00119377374241289
105	4.67E-04	3.06E-105	4.09E-14
106	2.18E-04	2.14E-22	0.00143827310126154
107	0.0022701886872815	5.66E-05	0.00184534437798825
108	1.69E-06	1.95E-19	9.17E-04
109	2.05E-04	9.12E-17	0.00264495271428907
110	0.0023491231975022	1.35E-39	6.30E-04
111	0.0030400885749542	8.84E-04	0.00186804038340817
112	1.58E-04	5.63E-04	7.62E-04
113	3.73E-10	2.45E-44	0.00110194462602343
114	1.74E-04	7.83E-34	2.88E-04
115	0.0014801522728398	1.91E-10	0.00267757980127201
116	1.95E-07	1.34E-04	3.86E-04
117	3.37E-27	2.58E-17	0.00330583497898949

	q	r	s
118	2.21E-17	4.61E-84	8.26E-04
119	7.35E-09	8.26E-23	0.00486691649064415
120	7.58E-10	2.60E-04	0.00565889438139465
121	0.0142824281999041	4.55E-05	0.00173065818569374
122	1.66E-04	3.43E-04	4.29E-05
123	1.16E-43	1.96E-65	0.00367314997665507
124	4.19E-78	1.05E-95	9.18E-05
125	4.26E-04	4.04E-13	7.75E-04
126	0	0	0
127	0.00110204924329289	1.37E-10	0.00415112964283091
128	3.29E-09	4.90E-34	0.00174474300244359
129	4.03E-10	2.91E-04	0.00153602403674259
130	0	0	0
131	0.00635355111900175	8.74E-23	5.67E-11
132	3.00E-28	4.22E-04	2.94E-07
133	0	0	0
134	4.87E-10	4.70E-34	0.00495875199017843
135	2.01E-25	8.53E-35	0.00128560249182927
136	0.0014824425105657	0.00365280515317119	0.00343168918472004
137	1.88E-10	7.28E-04	0.00957914040584579
138	6.39E-04	1.46E-38	8.41E-04
139	2.77E-04	2.09E-56	0.00321704236647535

	q	r	s
140	9.54E-05	4.47E-42	8.99E-05
141	3.74E-04	2.86E-84	2.75E-04
142	1.66E-04	2.08E-26	6.64E-04
143	4.92E-44	5.70E-29	0.0011937737424129
144	3.77E-07	8.31E-30	0.00486655290726183
145	2.64E-26	2.61E-12	0.00385680746867212
146	4.01E-84	4.56E-55	0.00110194499299652
147	4.53E-04	2.44E-05	0.0013280873898329
148	6.91E-26	2.91E-35	0.00174474623891116
149	3.40E-14	1.98E-16	0.00385680747545388
150	0.0013679644837195	5.08E-20	8.59E-04
151	7.17E-04	1.54E-04	0.00164987497031763
152	4.26E-49	1.67E-39	1.84E-04
153	4.57E-38	1.10E-23	0.00146925999066203
154	3.73E-05	1.76E-27	0.00281000259573987
155	3.45E-05	1.69E-04	9.03E-04
156	6.64E-05	2.22E-56	0.00140399072229868
157	2.39E-41	7.74E-38	0.0015610887400784
158	0.0014347191009199	2.18E-07	0.00299715112917363
159	5.54E-04	1.02E-35	9.25E-04
160	0.0017196719364407	5.44E-04	1.36E-05
161	0.0017861067280595	2.93E-41	0.00246871292392683

	q	r	s
162	1.32E-04	1.07E-64	6.05E-04
163	0.0021489952509276	4.37E-90	1.49E-15
164	1.85E-32	7.42E-49	6.43E-04
165	0.0034570793162963	9.42E-94	4.09E-13
166	7.82E-49	2.68E-29	3.67E-04
167	0.0016817852473849	4.04E-109	3.65E-08
168	4.07E-66	1.42E-84	9.18E-04
169	0.002525491244871	2.11E-39	0.00357861093323022
170	0.0016186807001465	6.65E-49	0.0065818976853605
171	0.0164368638993516	9.55E-13	7.49E-06
172	0.00214215784116607	7.20E-28	0.00175146613655301
173	6.72E-04	9.53E-29	0.00209409342295267
174	0	0	0
175	1.14E-21	5.38E-66	6.43E-04
176	0	0	0
177	0	0	0
178	0	0	0
179	0.0232374381120533	2.00E-13	0.00112923903996145
180	0	0	0
181	6.40E-42	3.52E-05	8.36E-49
182	8.47E-04	3.50E-04	9.68E-55
183	0	0	0

	q	r	s
184	0	0	0
185	0	0	0
186	0	0	0
187	0.0019792559696228	1.19E-11	9.93E-04
188	0.0308051268289229	3.01E-21	0.00232351593364892
189	0	0	0
190	0	0	0

Log Prob => -239038.55238082897

Appendix 2: HMM program results for the English text

HMM parsing on English text

No. of observations from file = 50000
No. of states = 2
No. of observations = 27
Iteration = 100

Start Probabilities

q	r
0.999999999999999	1.07E-15

Transition Probabilities

	q	r
q	0.224583895172636	0.77541610482723
r	0.709229231887198	0.290770768113053

Emission Probabilities

	q	r
a	0.176598231360123	0.0939315356517259
b	2.96E-12	0.0227466631765767
c	0.00133177077919124	0.0483726664748064
d	0.00403692431521916	0.0526764060366519
e	0.201672329581502	4.78E-16
f	1.41E-22	0.0466421477312508
g	0.00471250371119518	0.0201595377731672

	q	r
h	1.13E-05	0.0680381522273984
i	0.126060906030029	4.42E-12
j	7.05E-18	0.00298693556899635
k	0.00115546233898728	0.00468724706582281
l	1.26E-05	0.068075260424256
m	4.68E-19	0.0373749886582108
n	4.39E-11	0.106151094796516
o	0.128782247292518	1.11E-10
p	0.00594229848285901	0.0296421394503094
q	2.06E-47	0.00183811419630545
r	3.30E-19	0.0941267644691416
s	1.39E-05	0.101619700422603
t	0.0288288025406763	0.112983364839876
u	0.0433321276755627	3.99E-10
v	5.39E-38	0.0173854967733891
w	1.75E-11	0.0216744298821206
x	5.33E-28	0.00268058320294545
y	0.015438465978242	0.00904688873322159
z	1.39E-09	0.00202958315719238
space	0.262070081748743	0.0351302987728529

Log Prob => -198637.03686732217

No. of observations from file = 50000
No. of states = 3
No. of observations = 27
Iteration = 100

Start Probabilities

q	r	s
0.000000000000	1.000000000000000000	0.000000000000

Transition Probabilities

	q	r	s
q	0.13533663202	0.8646467711752	1.66E-05
r	0.50662789237	0.0515983099995569	0.441773797634675
s	0.64826047792	0.00153714039511984	0.35020238168674

Emission Probabilities

	q	r	s
a	0.233082287257705	0.113384931349495	0.00229656270433539
b	3.42E-12	0.0285410595973251	0.00651971056165521
c	3.32E-09	0.0558188268181198	0.0236337691163899
d	3.23E-05	0.0353654275910999	0.0681167057957474
e	0.145871330912725	8.81E-19	0.15822892948082
f	1.69E-16	0.0615467139798238	0.00892221372669597
g	3.96E-04	0.01691177722572	0.0266595318651568
h	4.85E-10	6.81E-05	0.144987346087114

	q	r	s
i	0.144717301753844	1.60E-18	0.0126333748798897
j	6.44E-21	0.00307262765993179	0.00184920619259806
k	7.13E-04	0.00340459093333521	0.00609065013749126
l	2.05E-12	0.0624424199441864	0.0533273414871314
m	6.89E-10	0.0436861920562881	0.0154332091548313
n	4.14E-06	0.152367930224929	0.00221366920129921
o	0.143276401821912	7.75E-17	0.0202624179993738
p	0.0029029241160618	0.0325251156488149	0.0222723729929628
q	1.07E-30	0.00246744241299992	2.90E-04
r	3.16E-08	0.0994736687929252	0.0543811110057978
s	0.0018086236473623	0.101437264324402	0.0645814181697098
t	9.22E-08	0.127038036428948	0.110264298382864
u	0.0509029006462384	1.34E-04	0.00227995084776153
v	1.89E-30	0.022069290792276	0.00460790046818679
0	3.53E-06	0.0260267454019291	0.00792600661910442
x	6.60E-09	0.00388578873485733	1.73E-34
y	0.0050437637459929	0.00505819826267009	0.0338291177595856
z	2.25E-04	0.00242891950022893	3.93E-04
space	0.27102021648372	8.45E-04	0.148000620332349

Log Prob => -193690.1452632959